

Parallel Performance Improvement

Performance Analysis

The Parallel Performance Improvement (PPI) effort is a collection of projects that are developing ways to improve the efficiency of parallel programs. We are investigating both performance analysis tools and system-level techniques for improving performance.

Optimizing the performance of a parallel program requires attention to CPU utilization, memory subsystem utilization, communication efficiency, and other areas. The Lawrence Livermore National Laboratory (LLNL) has developed tools to help developers investigate a variety of performance parameters.

The mpiP tool (Vetter and McCracken, 2001), for example, shows users how communication performance scales with increasing numbers of processors, and it offers information at varying levels of detail: categories of MPI functions (e.g., collective operations); specific MPI functions (e.g., all uses of MPI_Allreduce); and specific call sites (e.g., a call to MPI_Allreduce at a particular line of a particular file). These levels of detail enable users to see a general overview of communication performance and then zoom in to find specific problem areas. The mpiP tool gathers data from a whole program run without significantly slowing its execution, even when the program executes on 1000 processors or more (Figure 1).

Another tool, MPX (May, 2001), gathers statistics from CPU hardware performance counters (<http://www.llnl.gov/CASC/mpx>). Performance data (such as requests to load data, cache misses, and floating point operations) can help programmers tune their codes to improve cache utilization or balance workload among processors. Unfortunately, hardware counters have somewhat limited functionality, since

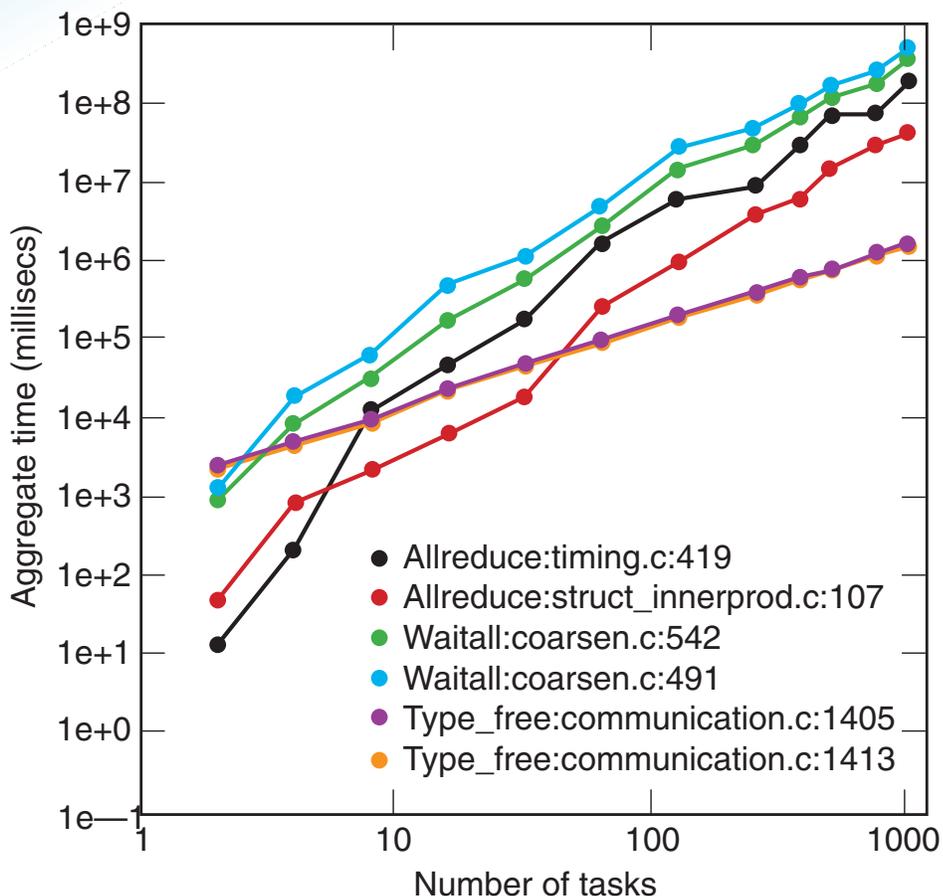


Figure 1. The mpiP tool shows how much time various MPI calls take as the number of tasks increases. Steep increases indicate possible scaling problems.

each counter is often designed to measure only a subset of the countable events on a CPU. Often, a combination of measurements is required to gain performance insights; for example, estimating cache utilization requires both the number of cache misses and the number of load requests. When hardware limitations prevent concurrent counting of hardware events, MPX uses time slicing, which directs the hardware registers to measure each event type in turn for a period of time. Using these measurements, MPX computes an accurate estimate of the frequency

of each event over a specified measurement period. MPX technology has been integrated into the Performance API toolkit, developed at the University of Tennessee (PAPI, <http://icl.cs.utk.edu/projects/papi>). Tool Gear is software infrastructure that provides common services to tools like MPX and mpiP (<http://www.llnl.gov/ascii/projects/asde/toolgear.html>). These services include source code navigation, dynamic instrumentation, data collection, and data display. Analysis tools focus on gathering data, and Tool Gear

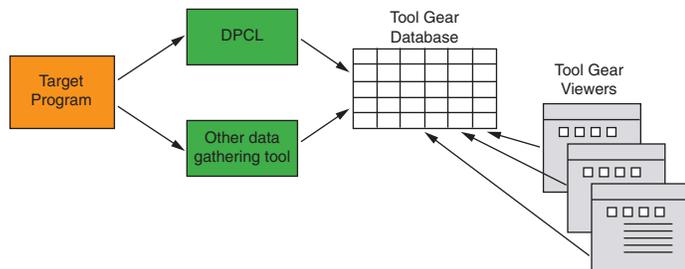


Figure 2. Tool Gear infrastructure gathers performance data from programs through the IBM Dynamic Probe Class Library (DPCL) or by other means and deposits it in a database. Customized viewers present this data in various ways and associate it with source code.

focuses on managing and presenting it, and also on user interactions (Figure 2). Tool Gear uses IBM's Dynamic Probe Class Library (DPCL), which lets tools insert instrumentation into a running program without the need for re-compilation or re-linking. Together, Tool Gear and DPCL offer a flexible combination of features that simplify the process of creating sophisticated performance analysis and debugging tools.

LLNL has used these and other tools (such as the Sphinx parallel microbenchmark suite, <http://www.llnl.gov/CASC/sphinx/sphinx.html>) to gather a wealth of data on the performance characteristics of several scientific applications.

Performance Tuning

The other goal of PPI is to develop system-level techniques for improving the performance of applications. Our main effort in this area is the Memory Wall project. This project is examining patterns of memory accesses that programs generate, and it is developing techniques for handling these accesses more efficiently within memory subsystems. While standard caching methods work well when programs access the same data items repeatedly or in strict sequential order, many scientific codes access data in strided patterns or in other complex ways. The Memory Wall project has developed tools to measure and characterize the memory access patterns of a program (T. Mohan, et. al, 2001 and E. Parker, et. al, 2001), and it is also investigating the use of dynamic

access optimization (DAO) to improve efficiency for these access patterns. DAO, which was first developed at the University of Virginia and the University of Utah, uses indirection to change a program's access patterns at runtime into ones that exhibit better spatial locality so that the memory system can handle them more efficiently. LLNL is collaborating with the developers of DAO and with SRC Computers to extend these techniques to shared memory parallel computers (Figure 3).

Collaborations

LLNL works with many academic partners and vendor institutions on its performance analysis and tuning efforts. We currently fund academic partners at the Universities of Maryland, Oregon, Utah, and Wisconsin, and at Portland State University. Vendor partners include IBM, MPI Software Technology, KAI, Pallas, and Etnus. LLNL also has many ongoing collaborations with other major universities, vendors, and national laboratories. LLNL is a participant in the Performance Evaluation Research Center, funded by the Department of Energy's Scientific Discovery through Advanced Computing (SciDAC) initiative.

Funding

Individual projects in PPI receive funding from a variety of sources, including ASCI Simulation Development

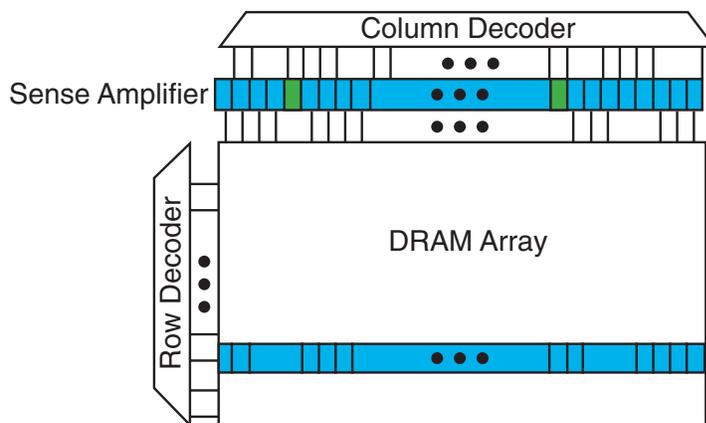


Figure 3. Dynamic Access Ordering (DAO) can group requests for data in the same row of a DRAM array so that an access to a single row can service multiple requests.

Environments program (ASDE), the Department of Energy's Scientific Discovery through Advanced Computing (SciDAC) program, and LLNL's Laboratory Directed Research and Development (LDRD) program.

References

- J.S. Vetter and M.O. McCracken, "Statistical Scalability Analysis of Communication Operations in Distributed Applications," Proc. ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPOPP), 2001.
- J.M. May, "MPX: Software for Multiplexing Hardware Performance Counters in Multithreaded Programs," Int. Parallel & Distr. Processing Sym. (IPDPS), 2001.
- T. Mohan, B.R. de Supinski, S.A. McKee, F. Mueller, and A. Yoo, "Dynamic Detection of Streams in Memory References," SC 2001, Denver, CO, November 10-16, 2001.
- E. Parker, B.R. de Supinski, and D.J. Quinlan, "Measuring the Regularity of Array References," SC 2001, Denver, CO, November 10-16, 2001.

More information

John May, johnmay@llnl.gov (PPI Project Leader); Bronis de Supinski, bronis@llnl.gov (Memory Wall Project Leader); or Jeffrey Vetter, jv@llnl.gov (ASDE Tools Project Leader)